Package: infomap (via r-universe)

November 1, 2026

Title Network Clustering with the Map Equation

Config/x-release-please-start-version marker

Version 2.16.0

Config/x-release-please-end marker

Author Daniel Edler [aut, cre], Anton Holmgren [aut], Martin Rosvall [aut], mapequation [cph]

Maintainer Daniel Edler <daniel.edler@umu.se>

Description R bindings for the Infomap network clustering algorithm. Infomap decomposes a network into modules by optimally compressing a description of information flows on the network using the Map Equation. Provides a high-level R6 facade over the SWIG-generated bindings, with idiomatic R helpers for inspecting the modular hierarchy, exporting to data frames, and integrating with the 'igraph' package.

License GPL-3

URL https://mapequation.org/infomap/, https://mapequation.r-universe.dev/infomap

BugReports https://github.com/mapequation/infomap/issues

Encoding UTF-8

SystemRequirements C++17

Depends R (>= 4.0)

Imports methods, R6

Suggests igraph, knitr, rmarkdown, testthat (>= 3.0.0), tibble

VignetteBuilder knitr

Config/testthat/edition 3

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Repository https://mapequation.r-universe.dev

Date/Publication 2026-11-01 15:29:33 UTC

RemoteUrl https://github.com/mapequation/infomap

RemoteRef HEAD

RemoteSha 39e7876b9f9fb6652c38c17879c801afe936c123

RemoteSubdir interfaces/R/infomap

Contents

as.data.frame.Infomap 2
as_communities 3
cluster_infomap 4
cluster_infomap_multilayer 6
construct_args 7
Infomap 8
infomap_options 20
main 24
multilayer_node 25

Index **26**

`as.data.frame.Infomap` *Coerce Infomap results to a data.frame*

Description

Returns one row per leaf node in the partitioned tree. Mirrors Python's `Infomap.get_dataframe()`.

Usage

```
## S3 method for class 'Infomap'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  states = TRUE,
  level = 1L,
  tibble = FALSE,
  ...,
  depth_level = NULL
)
```

Arguments

x	An Infomap instance after run() has been called.
row.names	Standard as.data.frame argument; ignored.
optional	Standard as.data.frame argument; ignored.
states	If TRUE, return one row per state node (for higher-order networks); otherwise merge state nodes with the same physical id within a module. Default TRUE.
level	Level of the hierarchy used for the module_id column. 1 gives top-level modules, -1 the bottom level. Default 1.
tibble	If TRUE, return a tibble (requires the tibble package). Default FALSE returns a plain data.frame so the return type is independent of installed packages.
...	Unused.
depth_level	Deprecated spelling of level; leaves the infomap R surface in 3.0.

Value

A data.frame (or a tibble when tibble = TRUE) with columns state_id, node_id, module_id, flow, name, (for multilayer networks) layer_id and (for higher-order networks) state_name. The name column always carries the physical node name; state_name carries the per-state-node name, falling back to the physical name when a state node is unnamed.

Examples

```
im <- Infomap(silent = TRUE)
im$add_links(list(c(1, 2), c(2, 3), c(3, 1), c(3, 4), c(4, 5), c(5, 6), c(6, 4)))
im$run()
as.data.frame(im)
```

---

as_communities *Convert an Infomap result to igraph communities*

Description

Converts an infomap_result created from an igraph graph to an igraph communities object.

Usage

```
as_communities(x, graph, ...)

## S3 method for class 'infomap_result'
as_communities(x, graph, ...)
```

Arguments

x	An infomap_result object.
graph	The same igraph graph passed to cluster_infomap().
...	Unused.

Value

An igraph `communities` object.

Examples

```
if (requireNamespace("igraph", quietly = TRUE)) {
  graph <- igraph::make_graph(c(1, 2, 2, 3, 3, 1), directed = FALSE)
  result <- cluster_infomap(graph, silent = TRUE)
  as_communities(result, graph)
}
```

`cluster_infomap` *Cluster a network with Infomap*

Description

High-level helper for the common case: pass an edge list, run Infomap, and get a compact result object with node assignments and summary fields.

Usage

```
cluster_infomap(
  edges,
  weight = NULL,
  e.weights = NULL,
  v.weights = NULL,
  nb.trials = NULL,
  args = NULL,
  opts = NULL,
  tibble = FALSE,
  ...
)

## S3 method for class 'infomap_result'
print(x, ...)

## S3 method for class 'infomap_result'
summary(object, ...)

## S3 method for class 'infomap_result'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`edges`	A data frame, matrix, or igraph graph.

weight	Edge weight column. For data frames, use a column name or numeric column index. For matrices, use a numeric column index. Set FALSE to ignore weights. For igraph graphs, use the edge attribute name, NULL to use "weight" when present, or FALSE to ignore igraph edge weights.
e.weights	Alias for weight, provided for familiarity with igraph::cluster_infomap(). For igraph inputs, this can be an edge attribute name or a numeric vector with one value per edge. Do not pass both weight and e.weights.
v.weights	Vertex weights are not supported by this high-level helper yet.
nb.trials	Alias for the Infomap num_trials option, provided for familiarity with igraph::cluster_infomap(). Do not pass both nb.trials and num_trials.
args	Optional raw CLI argument string passed to Infomap().
opts	Optional options list from infomap_options().
tibble	If TRUE, return nodes as a tibble (requires the tibble package). Default FALSE returns a plain data frame.
...	Named option overrides forwarded to Infomap().
x	An infomap_result object.
object	An infomap_result object.
row.names	Standard as.data.frame argument; ignored.
optional	Standard as.data.frame argument; ignored.

Details

For data.frame and matrix inputs, the first two columns are treated as source and target node ids. Node ids must be numeric or integer and are passed through to the low-level Infomap() API unchanged. If weight = NULL, a third edge-list column is used as weight when present; otherwise all links get weight 1.

For igraph inputs, this function delegates to Infomap$add_igraph(). Infomap results use R igraph's 1-indexed vertex ids; the mapping field maps those ids back to vertex names when the graph has names.

Value

An object of class "infomap_result" with fields nodes, modules, codelength, num_top_modules, model, and related summary fields.

Examples

```
edges <- data.frame(
  source = c(1, 1, 2, 3, 4, 4, 5),
  target = c(2, 3, 3, 4, 5, 6, 6)
)
result <- cluster_infomap(edges, silent = TRUE, num_trials = 5)
result$codelength
result$modules
as.data.frame(result)
```

`cluster_infomap_multilayer` *Cluster a multilayer network with Infomap*

Description

High-level helper for multilayer networks: pass a single data frame (or matrix / tibble) of multilayer edges, run Infomap, and get an `infomap_result` whose `nodes` data frame includes a `layer_id` column.

Usage

```
cluster_infomap_multilayer(
  multilayer_edges,
  weight = NULL,
  args = NULL,
  opts = NULL,
  tibble = FALSE,
  ...
)
```

Arguments

`multilayer_edges`	A data frame, tibble, or matrix of multilayer edges in one of the formats above.
`weight`	Edge weight column name or numeric column index. Set `FALSE` to ignore weights. If `NULL` (default), uses a data-frame column named `weight` when present, then falls back to a single non-format data-frame column, then to weight 1.
`args`	Optional raw CLI argument string passed to `Infomap()`.
`opts`	Optional options list from `infomap_options()`.
`tibble`	If `TRUE`, return `nodes` as a tibble (requires the `tibble` package). Default `FALSE` returns a plain data frame.
`...`	Named option overrides forwarded to `Infomap()`.

Details

Two input formats are recognised by column name (case-sensitive):

- **Full multilayer**: columns `layer_from`, `node_from`, `layer_to`, `node_to`, plus an optional weight column. Each row is an arbitrary `(layer, node) -> (layer, node)` link, routed via `Infomap$add_multilayer_links()`. Use this format to mix intra-layer links (same `layer_from`/`layer_to`) and inter-layer links (same `node_from`/`node_to` across different layers) in a single edge list.

- **Intra-layer only**: columns `layer`, `node_from`, `node_to`, plus an optional weight column. Every row is a link within one layer, routed via `Infomap$add_multilayer_intra_links()`. With no explicit inter-layer links, Infomap couples layers via `multilayer_relax_rate` (default 0.15).

Data-frame columns may appear in any order. Matrix inputs are interpreted positionally with 3 columns for unweighted intra-layer links and 4 or 5 columns for full multilayer links. Use a data frame for weighted intra-layer-only inputs.

For more exotic setups (e.g. weighted inter-layer couplings via `add_multilayer_inter_links()`), use the lower-level `Infomap()` R6 API directly.

Value

An object of class `"infomap_result"`. The `nodes` field includes a `layer_id` column.

Examples

```
# Intra-layer only: two triangles, one per layer.
intra <- data.frame(
  layer     = c(1, 1, 1, 2, 2, 2),
  node_from = c(1, 2, 3, 1, 2, 3),
  node_to   = c(2, 3, 1, 2, 3, 1)
)
result <- cluster_infomap_multilayer(intra, silent = TRUE, num_trials = 3)
result$codelength
as.data.frame(result)

# Full multilayer: intra-layer triangles + inter-layer couplings.
edges <- data.frame(
  layer_from = c(1, 1, 1, 2, 2, 2, 1, 1, 1),
  node_from  = c(1, 2, 3, 1, 2, 3, 1, 2, 3),
  layer_to   = c(1, 1, 1, 2, 2, 2, 2, 2, 2),
  node_to    = c(2, 3, 1, 2, 3, 1, 1, 2, 3),
  weight     = c(1, 1, 1, 1, 1, 1, 0.5, 0.5, 0.5)
)
result <- cluster_infomap_multilayer(edges, silent = TRUE, num_trials = 3)
result$num_top_modules
```

`construct_args` *Render an Infomap options list to a CLI argument string*

Description

This is exported for advanced use; most callers should pass options directly to `Infomap()` or `Infomap$run()`.

Usage

```
construct_args(args = NULL, opts = NULL)
```

Arguments

args	Optional raw argument string to prepend.
opts	Options list from infomap_options() (or NULL).

Value

A single character string of CLI arguments.

Values containing whitespace

Options reach the engine as one whitespace-separated argument string, which has no quoting and no escaping, so a value containing a space cannot survive the trip – quotes are not stripped, and --out-name "my run" would set the name to the literal "my. Such a value is refused, naming the option, rather than truncated silently. The check runs when options are rendered, not when they are built, so infomap_options(out_name = "my run") returns a list and the error comes from construct_args() – or from Infomap() / $run(), which render internally. The Python binding differs here: it validates when its Options object is constructed. This applies to every path and free-string option, and is a limitation of the engine boundary shared with the Python and JavaScript bindings and the command line. Rename the file, or use a directory whose path has no spaces. A raw args string is passed through unvalidated.

Infomap *Infomap network clustering*

Description

Constructor for an Infomap network clustering instance. Mirrors the Python Infomap class in shape, with named arguments for every CLI flag and active bindings for read-only properties.

R6 class encapsulating an Infomap clustering session. Most users should call the Infomap() wrapper rather than constructing this class directly. The generator is exported for subclassing and method introspection.

Usage

```
Infomap(args = NULL, opts = NULL, ...)
```

Arguments

args	Optional raw CLI argument string to prepend.
opts	Optional options list from infomap_options().
...	Named option overrides forwarded to infomap_options().

Details

Build a network by calling `add_node()` / `add_link()` (or pass an existing igraph object to `add_igraph()`), then run with `run()`. Inspect results via the active bindings (`codelength`, `modules`, `num_top_modules`, ...) or use `as.data.frame()` for a tidy node-level frame.

Returned objects have R6 class `"Infomap"`; the underlying R6ClassGenerator is also exported as `InfomapClass` for advanced use (subclassing, method introspection).

Methods are grouped here as input (build the network), run (optimise the partition), igraph integration, results (read out module assignments and node attributes), and writers (export to .tree / .clu / etc).

Value

An object of R6 class `"Infomap"`.

Active bindings

`swig` The underlying SWIG-generated InfomapWrapper handle.

`network` The underlying Network reference.

An escape hatch, not the supported build surface. Mutating through it bypasses the wrapper, so the run's input file identity is not dropped and the published artifacts keep naming a file that no longer describes the network (#1026). Use the `add_*` and `remove_*` methods on this object.

The handle carries a reference to its owner, so it stays valid for as long as it is reachable. Without it the handle was a non-owning pointer into the C++ Network held by the SWIG wrapper: once this object became unreachable the wrapper was finalised and reads through the handle touched freed memory, returning plausible wrong values (numNodes 11 for a three-node network) or aborting R (#900).

`codelength` Total (hierarchical) codelength.

`codelengths` Codelength of each trial.

`num_top_modules` Number of top modules in the tree.

`num_non_trivial_top_modules` Number of non-trivial top modules.

`num_levels` Depth of the hierarchical tree.

`max_tree_depth` Deprecated alias of `num_levels`; leaves the infomap R surface in 3.0.

`num_leaf_nodes` Number of leaf nodes in the tree.

`num_nodes` Number of nodes (state nodes for higher-order networks).

`num_physical_nodes` Number of physical nodes.

`num_links` Number of links.

`have_memory` `TRUE` if this is a state/multilayer network.

`index_codelength` Index codelength (top-level part of the codelength).

`module_codelength` Module codelength (within-module part).

`hierarchical_codelength` Hierarchical codelength.

`one_level_codelength` One-level codelength baseline.

`relative_codelength_savings` Relative codelength savings.

`entropy_rate` Entropy rate of the network.

`max_entropy` Maximum possible entropy.

`bipartite_start_id` Get or set the bipartite start id.

`modules` Top-level module assignment per node (named integer vector).

`multilevel_modules` List of integer paths for each node through the module hierarchy.

`nodes` Per-state-node attributes (state id, physical id, module, flow, optional layer id).

`physical_nodes` Per-physical-node attributes (state nodes merged within a module).

`links` Per-link source, target, weight and flow as a `data.frame`.

`flow_links` Per-link source, target and flow as a `data.frame`.

`names` Named character vector of all assigned node names.

Methods

Public methods:

- `InfomapClass$new()`
- `InfomapClass$read_file()`
- `InfomapClass$add_node()`
- `InfomapClass$add_nodes()`
- `InfomapClass$add_state_node()`
- `InfomapClass$add_state_nodes()`
- `InfomapClass$set_name()`
- `InfomapClass$set_names()`
- `InfomapClass$add_link()`
- `InfomapClass$add_links()`
- `InfomapClass$remove_link()`
- `InfomapClass$remove_links()`
- `InfomapClass$add_multilayer_link()`
- `InfomapClass$add_multilayer_intra_link()`
- `InfomapClass$add_multilayer_intra_links()`
- `InfomapClass$add_multilayer_inter_link()`
- `InfomapClass$add_multilayer_inter_links()`
- `InfomapClass$add_multilayer_links()`
- `InfomapClass$set_meta_data()`
- `InfomapClass$run()`
- `InfomapClass$get_bipartite_start_id()`
- `InfomapClass$set_bipartite_start_id()`
- `InfomapClass$print()`
- `InfomapClass$add_igraph()`
- `InfomapClass$as_communities()`
- `InfomapClass$get_modules()`

- InfomapClass$get_multilevel_modules()
- InfomapClass$get_nodes()
- InfomapClass$get_links()
- InfomapClass$get_name()
- InfomapClass$get_names()
- InfomapClass$get_state_name()
- InfomapClass$get_state_names()
- InfomapClass$write_clu()
- InfomapClass$write_tree()
- InfomapClass$write_flow_tree()
- InfomapClass$write_newick()
- InfomapClass$write_json()
- InfomapClass$write_csv()
- InfomapClass$write_pajek()
- InfomapClass$write_state_network()
- InfomapClass$write()

Method `new()`**:** Create a new Infomap instance.

Usage:

```
InfomapClass$new(args = NULL, opts = NULL, ...)
```

Arguments:

`args` Optional raw CLI argument string to prepend.

`opts` Optional options list from `infomap_options()`.

`...` Named option overrides forwarded to `infomap_options()`.

Method `read_file()`**:** Read network data from file.

Usage:

```
InfomapClass$read_file(filename, accumulate = TRUE)
```

Arguments:

`filename` Path to a network file (`.net`, `.txt`, `.tree`, etc.).

`accumulate` If `TRUE` (default), accumulate to existing nodes/links.

Method `add_node()`**:** Add a node.

Usage:

```
InfomapClass$add_node(node_id, name = NULL, teleportation_weight = NULL)
```

Arguments:

`node_id` Integer node id.

`name` Optional node name.

`teleportation_weight` Optional teleportation weight.

Method `add_nodes()`**:** Add many nodes at once.

Usage:

```
InfomapClass$add_nodes(nodes)
```

Arguments:

`nodes` A list of integer node ids, or a list of vectors of the form `c(node_id, name, teleportation_weight)` (last two optional), or a named list/vector mapping `node_id` to `name` or to `c(name, teleportation_weight)`.

Method `add_state_node()`**:** Add a state node.

Usage:

```
InfomapClass$add_state_node(state_id, node_id, name = NULL)
```

Arguments:

`state_id` Integer state node id.

`node_id` Integer physical node id.

`name` Optional name of the state node itself, as opposed to the physical node.

Method `add_state_nodes()`**:** Add many state nodes at once.

Usage:

```
InfomapClass$add_state_nodes(state_nodes)
```

Arguments:

`state_nodes` A list of `c(state_id, node_id)` or `c(state_id, node_id, name)` vectors, or a named list/vector mapping `state_id` to `node_id`.

Method `set_name()`**:** Set the name of a node.

Usage:

```
InfomapClass$set_name(node_id, name)
```

Arguments:

`node_id` Integer node id.

`name` Node name (character). `NULL` clears the name.

Method `set_names()`**:** Set names for several nodes at once.

Usage:

```
InfomapClass$set_names(mapping)
```

Arguments:

`mapping` A list of `c(node_id, name)` vectors, or a named list/vector mapping `node_id` to `name`.

Method `add_link()`**:** Add a link.

Usage:

```
InfomapClass$add_link(source_id, target_id, weight = 1)
```

Arguments:

`source_id` Source node id.

`target_id` Target node id.

`weight` Link weight.

Method `add_links()`**:** Add many links at once.

Usage:

```
InfomapClass$add_links(links)
```

Arguments:

`links` A list of vectors of the form `c(source, target, weight)` (weight optional), or a 2- or 3-column matrix / data.frame whose first two columns are integer/numeric node ids and (optionally) third column is numeric weight.

Method `remove_link()`**:** Remove a link.

Usage:

```
InfomapClass$remove_link(source_id, target_id)
```

Arguments:

`source_id` Source node id.

`target_id` Target node id.

Method `remove_links()`**:** Remove many links at once.

Usage:

```
InfomapClass$remove_links(links)
```

Arguments:

`links` A list of `c(source, target)` vectors.

Method `add_multilayer_link()`**:** Add a multilayer link.

Usage:

```
InfomapClass$add_multilayer_link(
  source_multilayer_node,
  target_multilayer_node,
  weight = 1
)
```

Arguments:

`source_multilayer_node` A `c(layer_id, node_id)` vector or the result of `multilayer_node()`.

`target_multilayer_node` A `c(layer_id, node_id)` vector or the result of `multilayer_node()`.

`weight` Link weight.

Method `add_multilayer_intra_link()`**:** Add an intra-layer link in a multilayer network.

Usage:

```
InfomapClass$add_multilayer_intra_link(
  layer_id,
  source_node_id,
  target_node_id,
  weight = 1
)
```

Arguments:

`layer_id` Integer layer id.

`source_node_id` Source node id.

`target_node_id` Target node id.

`weight` Link weight.

Method `add_multilayer_intra_links()`: Add many intra-layer links in a multilayer network.

Usage:

```
InfomapClass$add_multilayer_intra_links(links)
```

Arguments:

`links` A list of vectors of the form `c(layer, source_node, target_node, weight)` (weight optional), or a 3- or 4-column matrix / data.frame.

Method `add_multilayer_inter_link()`: Add an inter-layer link in a multilayer network.

Usage:

```
InfomapClass$add_multilayer_inter_link(
  source_layer_id,
  node_id,
  target_layer_id,
  weight = 1
)
```

Arguments:

`source_layer_id` Source layer id.

`node_id` Physical node id (same in both layers).

`target_layer_id` Target layer id.

`weight` Link weight.

Method `add_multilayer_inter_links()`: Add many inter-layer links in a multilayer network.

Usage:

```
InfomapClass$add_multilayer_inter_links(links)
```

Arguments:

`links` A list of vectors of the form `c(source_layer, node, target_layer, weight)` (weight optional), or a 3- or 4-column matrix / data.frame.

Method `add_multilayer_links()`: Add many multilayer links at once.

Usage:

```
InfomapClass$add_multilayer_links(links)
```

Arguments:

`links` A list whose entries are `list(source_multilayer_node, target_multilayer_node, weight)` (weight optional), or a 4- or 5-column matrix / data.frame of `source_layer`, `source_node`, `target_layer`, `target_node`, and optional `weight`.

Method `set_meta_data()`: Set meta data for a node.

Usage:

```
InfomapClass$set_meta_data(node_id, meta_category)
```

Arguments:

`node_id` Integer node id.

`meta_category` Integer meta category.

Method `run()`: Run Infomap.

Usage:

```
InfomapClass$run(args = NULL, opts = NULL, ...)
```

Arguments:

`args` Optional raw CLI argument string for this run only.

`opts` Optional `infomap_options()` result for this run only.

`...` Named option overrides for this run only.

Method `get_bipartite_start_id()`**:** Get the bipartite start id (the node id where the second node type starts).

Usage:

```
InfomapClass$get_bipartite_start_id()
```

Method `set_bipartite_start_id()`**:** Set the bipartite start id.

Usage:

```
InfomapClass$set_bipartite_start_id(start_id)
```

Arguments:

`start_id` Integer node id where the second node type starts.

Method `print()`**:** Print a short summary.

Usage:

```
InfomapClass$print(...)
```

Arguments:

`...` Unused.

Method `add_igraph()`**:** Import an igraph graph.

Usage:

```
InfomapClass$add_igraph(
  g,
  weight = "weight",
  phys_id = "phys_id",
  layer_id = "layer_id",
  multilayer_inter_intra_format = TRUE
)
```

Arguments:

`g` An igraph graph.

`weight` Edge attribute to use as link weight. Use `NULL` to use `"weight"` when present, or `FALSE` to ignore edge weights.

`phys_id` Vertex attribute holding physical node ids (state-node case).

`layer_id` Vertex attribute holding layer ids (multilayer case).

`multilayer_inter_intra_format` If `TRUE`, intra/inter format is used; diagonal links (different layer AND different physical node) trigger an error. Set to `FALSE` to allow `add_multilayer_link()` for arbitrary (layer, node) pairs.

Details: Translated from `interfaces/python/src/infomap/_networkx.py`.
Node id convention. Infomap result accessors report the numeric node ids used to build the network. For links added directly with `add_link()` or `add_links()`, those user-supplied ids are preserved. `add_igraph()` uses R igraph's 1-indexed vertex ids as state ids. Plain graph results therefore use those ids directly. State and multilayer graphs may use separate physical ids from `phys_id`; if those ids are labels, they are mapped to stable internal integers and returned as `attr(mapping, "phys_id")`. If the original igraph had vertex names (`V(g)$name`), the returned mapping recovers them.
If the graph is directed, `run()` will inject `--directed` unless the user has already chosen a flow model via `opts` or raw args.

Returns: Invisibly returns a named character vector mapping igraph vertex ids to the original igraph vertex names (or stringified vertex ids when `V(g)$name` is absent). For state or multilayer networks with non-numeric `phys_id` labels, the returned vector has a `"phys_id"` attribute mapping internal physical ids to original labels. Useful for joining Infomap results back to the original graph.

Method `as_communities()`**:** Convert the Infomap partition to an igraph `communities` object (from `igraph::make_clusters()`), compatible with `modularity()`, `membership()`, `plot()`.

Usage:

```
InfomapClass$as_communities(g)
```

Arguments:

`g` The same igraph graph passed to `add_igraph()`.

Method `get_modules()`**:** Get module assignment per leaf node.

Usage:

```
InfomapClass$get_modules(level = 1L, states = FALSE, depth_level = NULL)
```

Arguments:

`level` Level of the hierarchy used for the module id. `1` gives top-level modules, `-1` the bottom level.

`states` If `TRUE`, return one entry per state node (for higher-order networks); otherwise one per physical node.

`depth_level` Deprecated spelling of `level`; leaves the infomap R surface in 3.0.

Returns: A named integer vector mapping node id (or state id) to module id.

Method `get_multilevel_modules()`**:** Get the full module path for each leaf node.

Usage:

```
InfomapClass$get_multilevel_modules(states = FALSE)
```

Arguments:

`states` If `TRUE`, use state ids for higher-order networks.

Returns: A named list mapping node id (or state id) to an integer vector of module ids per level.

Method `get_nodes()`**:** Get per-leaf-node attributes (state id, physical id, module id, flow, optional layer id).

Usage:

```
InfomapClass$get_nodes(level = 1L, states = FALSE, depth_level = NULL)
```

Arguments:

`level` Level of the hierarchy used for the module id.

`states` If `TRUE`, return one row per state node.

`depth_level` Deprecated spelling of `level`; leaves the infomap R surface in 3.0.

Returns: A list of integer/numeric vectors.

Method `get_links()`**:** Get per-link weights and flow.

Usage:

```
InfomapClass$get_links()
```

Details: For ordinary networks added with `add_link()` or `add_links()`, `source` and `target` are the user-supplied node ids. For state and multilayer networks they are state ids. Before `run()`, `weight` reflects the input weights and `flow` reflects the core link-flow values currently stored by Infomap, usually zero.

Returns: A `data.frame` with columns `source`, `target`, `weight`, and `flow`.

Method `get_name()`**:** Look up a node's name.

Usage:

```
InfomapClass$get_name(node_id, default = NULL)
```

Arguments:

`node_id` Integer node id.

`default` Value returned when the node has no name.

Returns: Character (or `default`).

Method `get_names()`**:** Get all assigned node names.

Usage:

```
InfomapClass$get_names()
```

Returns: A named character vector mapping node id to name.

Method `get_state_name()`**:** Look up a state node's name.

Usage:

```
InfomapClass$get_state_name(state_id, default = NULL)
```

Arguments:

`state_id` Integer state node id.

`default` Value returned when the state node has no name.

Returns: Character (or `default`).

Method `get_state_names()`**:** Get all assigned state node names.

Usage:

```
InfomapClass$get_state_names()
```

Details: Populated for higher-order (state/memory) networks whose `*States` section names the state nodes; empty otherwise. Physical node names are available separately via `get_names()`.

Returns: A named character vector mapping state id to name.

Method `write_clu()`**:** Write the partition as a `.clu` file.

Usage:

```
InfomapClass$write_clu(
  filename,
  states = FALSE,
  level = 1L,
  depth_level = NULL
)
```

Arguments:

`filename` Output path.

`states` Whether to write state ids (default `FALSE`).

`level` Level of the hierarchy used for the module id.

`depth_level` Deprecated spelling of `level`; leaves the infomap R surface in 3.0.

Method `write_tree()`**:** Write the partition as a `.tree` file.

Usage:

```
InfomapClass$write_tree(filename, states = FALSE)
```

Arguments:

`filename` Output path.

`states` Whether to include state ids.

Method `write_flow_tree()`**:** Write the partition as a `.ftree` (flow-tree) file.

Usage:

```
InfomapClass$write_flow_tree(filename, states = FALSE)
```

Arguments:

`filename` Output path.

`states` Whether to include state ids.

Method `write_newick()`**:** Write the partition as a Newick `.nwk` file.

Usage:

```
InfomapClass$write_newick(filename, states = FALSE)
```

Arguments:

`filename` Output path.

`states` Whether to include state ids.

Method `write_json()`**:** Write the partition as a JSON file.

Usage:

```
InfomapClass$write_json(filename, states = FALSE)
```

Arguments:

`filename` Output path.

`states` Whether to include state ids.

Method `write_csv()`**:** Write the partition as a CSV file.

Usage:

```
InfomapClass$write_csv(filename, states = FALSE)
```

Arguments:

`filename` Output path.

`states` Whether to include state ids.

Method `write_pajek()`**:** Write the input network in Pajek `.net` format.

Usage:

```
InfomapClass$write_pajek(filename, flow = FALSE)
```

Arguments:

`filename` Output path.

`flow` Whether to write computed flow values per link.

Method `write_state_network()`**:** Write the state network.

Usage:

```
InfomapClass$write_state_network(filename)
```

Arguments:

`filename` Output path.

Method `write()`**:** Dispatch on file extension to the appropriate writer (`.tree`, `.ftree`, `.nwk`, `.clu`, `.json`, `.csv`, `.net`).

Usage:

```
InfomapClass$write(filename, ...)
```

Arguments:

`filename` Output path. Extension chooses the writer.

`...` Forwarded to the chosen writer.

Examples

```
# Two triangles joined by a bridge.
im <- Infomap(silent = TRUE, num_trials = 5)
im$add_links(list(c(1, 2), c(1, 3), c(2, 3),
                  c(3, 4),
                  c(4, 5), c(4, 6), c(5, 6)))
im$run()
im$num_top_modules
im$codelength
im$modules
```

`infomap_options` *Build a reusable Infomap options list*

Description

Returns a named list with one entry per Infomap CLI option. All arguments default to the Infomap-internal defaults, so callers only need to supply the options they want to override. The returned list is accepted by `Infomap()` and `Infomap$run()`.

Usage

```
infomap_options(...)
```

Arguments

`...`	Named option overrides. See **Options** below.

Details

Argument names mirror the Infomap CLI flags (with hyphens replaced by underscores) and the keyword arguments accepted by the Python interface.

Value

A named list of options.

Options

Input

`include_self_links` Deprecated. Self-links are included by default; use `no_self_links = TRUE` to exclude them.

`skip_adjust_bipartite_flow` Keep flow on bipartite nodes instead of distributing it to primary nodes.

`bipartite_teleportation` Use bipartite teleportation instead of the default two-step unipartite teleportation.

`weight_threshold` Ignore input links with weight below this threshold.

`no_self_links` Exclude self-links from the input network.

`node_limit` Read only nodes up to this node id and ignore links connected to higher node ids.

`matchable_multilayer_ids` Construct state ids from node ids and layer ids that stay comparable across networks. Set at least to the largest layer id among networks to match.

`cluster_data` Read an initial partition from a clu file or a hierarchy from a tree/ftree file. Tree input may use physical or state nodes for higher-order networks.

`assign_to_neighbouring_module` With –cluster-data, assign nodes missing module ids to a neighboring node's module when possible.

`meta_data` Read metadata to encode from a clu-format file.

`meta_data_rate` With –meta-data, set the metadata encoding rate. The default encodes metadata at each step.

`meta_data_unweighted` With –meta-data, encode metadata without weighting by node flow.

`no_infomap` Skip optimization. Use this to calculate codelength for –cluster-data or to print non-modular statistics.

Output

`out_name` Base name for output files, for example [out_directory]/[out-name].tree.

`no_file_output` Do not write output files.

`tree` Write the modular hierarchy to a tree file. Enabled by default when no other output format is selected.

`ftree` Write the modular hierarchy and aggregated links between nested modules to an ftree file. Used by Network Navigator.

`clu` Write top-level module ids for each node to a clu file.

`clu_level` With –clu or –output clu, write module ids at this depth from the root. Use -1 for bottom-level modules.

`output` Write selected output formats as a comma-separated list without spaces, e.g. -o clu,tree,ftree. Options: clu, tree, ftree, newick, json, csv, network, states, flow.

`hide_bipartite_nodes` Hide bipartite nodes in output by projecting the solution to primary nodes.

`print_all_trials` Write each trial to separate output files. Has effect only when –num-trials is greater than 1.

`no_overwrite` Fail with an output error if any target output file already exists. By default existing files are replaced.

`print_config_fingerprint` Print the canonical configuration fingerprint and exit. Deprecated: leaves the R surface in 3.0. A print-and-exit CLI diagnostic; run the infomap binary.

`timing_json` Write machine-readable run timing JSON to this path. Use - for stdout.

`summary_json` Write machine-readable final run summary JSON to this path. Use - for stdout.

`manifest_json` Write a machine-readable run manifest JSON to this path. Use - for stdout.

`memory_report` Include peak RSS and best-effort bytes per node/link estimates in timing JSON. Requires –timing-json.

`trial_offset` Global index of the first trial this process runs; trial i uses seed = base_seed + (trial_offset + i). Default 0 (single-process behavior).

`trial_results` Write this shard's per-trial results (codelengths, seeds, best-tree reference, fingerprints) as JSON to this path, for deterministic merging of distributed shard runs into a final solution.

`no_final_output` Skip writing this process's aggregate best result. Per-trial outputs and –trial-results are still written.

`verbosity_level` Increase console verbosity. Add more v flags to increase verbosity up to -vvv.

`silent` Suppress console output.

Algorithm

`two_level` Optimize a two-level partition instead of the default multi-level hierarchy.

`flow_model` Choose how Infomap derives flow from the input links. Options: undirected, directed, undirdir, outdirdir, rawdir, precomputed.

`directed` Treat input links as directed. Shorthand for –flow-model directed.

`recorded_teleportation` When teleportation is used to calculate flow, also record teleportation steps in the codelength.

`use_node_weights_as_flow` Use node weights from the API or Pajek node records as normalized node flow.

`to_nodes` Teleport to nodes instead of links. Uses uniform node weights unless node weights are provided.

`teleportation_probability` Set the probability of teleporting to a random node or link when calculating flow.

`max_flow_iterations` Cap the power iteration used to calculate flow (directed and regularized flow models) at this many iterations. 0 sets no cap: the iteration runs until –flow-tolerance is met or the change in flow stops falling.

`min_flow_iterations` Require at least this many power iterations before the flow calculation can converge, even if –flow-tolerance is already met.

`flow_tolerance` Convergence tolerance for the power iteration used to calculate flow. Iteration stops once the per-iteration change in flow drops to or below this value, after –min-flow-iterations have run, or once that change has stopped falling.

`regularized` Add a fully connected Bayesian prior network to reduce overfitting to missing links. Activates –recorded-teleportation.

`regularization_strength` Scale the relative strength of the Bayesian prior network used by –regularized.

`intra_regularization_strength` Scale the intra-layer part of the multilayer prior used by –regularized, relative to –regularization-strength. Set to 0 to regularize only between layers.

`inter_regularization_strength` Scale the inter-layer part of the multilayer prior used by –regularized, relative to –regularization-strength. Set to 0 to leave the layers uncoupled.

`multilayer_skip_absent_nodes` Treat a physical node's absence from a layer as real rather than unobserved, so the prior network of –regularized spans only the nodes each layer holds.

`entropy_corrected` Correct for negative entropy bias in small samples, especially solutions with many modules.

`entropy_correction_strength` Scale the default correction used by –entropy-corrected.

`markov_time` Scale link flow to change the cost of moving between modules. Higher values result in fewer modules.

`variable_markov_time` Vary Markov time locally to reduce overpartitioning in sparse areas while keeping higher resolution in dense areas.

`variable_markov_damping` With –variable-markov-time, set damping between local effective degree (0) and local entropy (1).

`variable_markov_min_scale` With –variable-markov-time, set the minimum local scale for zero-entropy nodes. Local Markov time is max scale divided by local scale.

`preferred_number_of_modules` Penalize solutions by how far their number of modules differs from this value.

`preferred_number_of_levels` Soft preference for the depth of the hierarchy. Steering to a shallower depth is reliable at a small codelength cost; deeper is best-effort, bounded by what the optimizer proposes. No-op with –two-level or strength 0.

`preferred_number_of_levels_strength` Scale the strength of –preferred-number-of-levels. 0 disables the preference; larger values increase the cost of deviating from the preferred depth.

`multilayer_relax_rate` Set the probability of relaxing from a state node to neighboring layers instead of staying in the current layer.

`multilayer_relax_limit` Limit relaxation to this many neighboring layer ids in each direction. Use a negative value to allow relaxation to any layer.

`multilayer_relax_limit_up` Limit relaxation upward to this many higher neighboring layer ids. Use a negative value to allow relaxation to any higher layer.

`multilayer_relax_limit_down` Limit relaxation downward to this many lower neighboring layer ids. Use a negative value to allow relaxation to any lower layer.

`multilayer_relax_by_jsd` Weight multilayer relaxation by out-link similarity measured with Jensen-Shannon divergence.

`multilayer_relax_to_self` On relaxation, link a state node to its own physical node in the target layer instead of spreading to its out-neighbors. Builds a smaller state network with the same flow as the default.

Accuracy

`seed` Set the random number generator seed for reproducible results.

`num_trials` Run this many independent trials and keep the best solution.

`core_loop_limit` Limit how many core loops try to move each node to the best module.

`core_level_limit` Limit how many times core loops are reapplied to the aggregated modular network to find larger structures. 0 means no limit.

`tune_iteration_limit` Limit the main iterations in the two-level partition algorithm. 0 means no limit.

`core_loop_codelength_threshold` Require at least this codelength improvement to accept a new solution in a core loop.

`tune_iteration_relative_threshold` Require each tune iteration to improve codelength by this fraction of the initial two-level codelength.

`fast_hierarchical_solution` Find top modules quickly. Use -FF to keep all fast levels. Use -FFF to skip recursive refinement.

`inner_parallelization` Experimental: use batched parallel node moves for coarse optimization. Performance gains are workload-dependent, often require a relaxed core-loop-codelength-threshold and low tune-iteration-limit, and may produce a different partition than serial optimization.

`parallel_trials` Run independent trials in parallel with OpenMP. –num-trials remains the total number of trials; the number of parallel workers follows the OpenMP thread count (e.g. OMP_NUM_THREADS), clamped to –num-trials. Peak memory scales with the worker count. Nested OpenMP and –inner-parallelization are disabled inside workers.

converge Treat the trial count as a cap and stop early once the best codelength has plateaued (no meaningful improvement over several consecutive trials). Runs trials serially; cannot be combined with parallel trials or distributed sharding. With no explicit trial count, a default cap is used.

num_threads Effective thread budget: 'auto' (resolve from –num-threads > INFOMAP_NUM_THREADS > SLURM_CPUS_PER_TASK > OMP_NUM_THREADS > cpuset > hardware), or a positive integer. 1 forces fully serial. Governs the recursive partition, parallel trials, and inner parallelization.

threads Alias for –num-threads. Deprecated: leaves the R surface in 3.0. Use num_threads; threads is a redundant alias of the same engine option.

prefer_modular_solution Prefer a modular solution even when one module gives a lower codelength.

num_random_moves Try this many random moves in each core loop to merge weakly connected nodes.

max_degree_for_random_moves Try random moves only for nodes with degree at most this value.

Examples

```
opts <- infomap_options(num_trials = 10, two_level = TRUE)
im <- Infomap(opts = opts, silent = TRUE)
```

---

main *Run Infomap from the command line*

Description

Parses arguments from commandArgs() (or the args parameter), runs Infomap, and exits. Useful for invoking the optimizer from Rscript: Rscript -e 'infomap::main()' --args <flags...>.

Usage

```
main(args = NULL)
```

Arguments

args	Character vector of CLI arguments (default: from commandArgs()).

Details

Each element of args is treated as a single token; spaces inside a token (e.g. in filenames) survive via shQuote. Pass tokens, not a single pre-joined string.

Value

Invisibly returns the resulting Infomap instance.

`multilayer_node` *Multilayer node helper*

Description

Convenience constructor for `c(layer_id, node_id)` integer pairs passed to multilayer-link methods.

Usage

```
multilayer_node(layer_id, node_id)
```

Arguments

`layer_id`	Integer layer id.
`node_id`	Integer node id.

Value

A length-2 integer vector with class `"multilayer_node"`.

Examples

```
src <- multilayer_node(1L, 2L)
tgt <- multilayer_node(1L, 3L)
```

Index

as.data.frame(), *9*
as.data.frame.Infomap, 2
as.data.frame.infomap_result
        *(*cluster_infomap*)*, 4
as_communities, 3

cluster_infomap, 4
cluster_infomap(), *3*
cluster_infomap_multilayer, 6
construct_args, 7

Infomap, 8
Infomap(), *5–8*, *20*
infomap_options, 20
infomap_options(), *5, 6*, *8*, *11*, *15*
InfomapClass *(*Infomap*)*, 8

main, 24
multilayer_node, 25
multilayer_node(), *13*

print.infomap_result *(*cluster_infomap*)*,
        4

summary.infomap_result
        *(*cluster_infomap*)*, 4